PCT WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 6 : G06F 15/20 — A1

(11) International Publication Number: WO 99/50756

(43) International Publication Date: 7 October 1999 (07.10.99)

(21) International Application Number: PCT/US99/06126

(22) International Filing Date: 26 March 1999 (26.03.99)

(30) Priority Data:
60/079,611 27 March 1998 (27.03.98) US

(71) Applicant *(for all designated States except US)*: INFOIMAGE INCORPORATED [US/US]; Suite 2310, 100 West Clarendon, Phoenix, AZ 85013 (US).

(72) Inventors; and
(75) Inventors/Applicants *(for US only)*: AANNESTAD, Bjorn [US/US]; 228 East Fremont, Tempe, AZ 85282 (US). STACK, John [US/US]; 740 West Coronado, Phoenix, AZ 85007 (US).

(74) Agent: POTE, Daniel, R.; Snell & Wilmer L.L.P., One Arizona Center, 400 East Van Buren, Phoenix, AZ 85004-0001 (US).

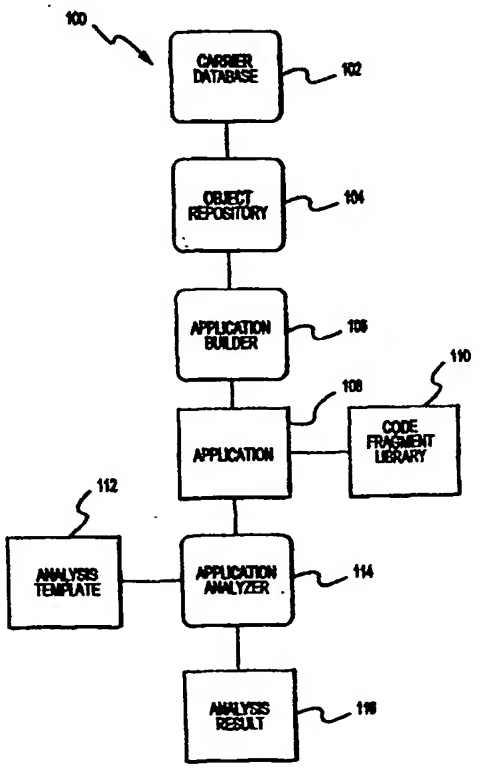
(81) Designated States: AE, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, CA, CH, CN, CU, CZ, DE, DK, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MD, MG, MK, MN, MW, MX, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, UA, UG, US, UZ, VN, YU, ZA, ZW, ARIPO patent (GH, GM, KE, LS, MW, SD, SL, SZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GW, ML, MR, NE, SN, TD, TG).

Published
With international search report.
Before the expiration of the time limit for amending the claims and to be republished in the event of the receipt of amendments.

(54) Title: METHODS AND APPARATUS FOR NETWORK APPLICATIONS USING OBJECT TOOLS

(57) Abstract

Methods and apparatus are provided for developing object-oriented network groupware applications (108) in substantially non object-oriented programming environments. Object-oriented architectural and naming standards are used in conjunction with object repositories (104), an application builder (106), an application analyzer (114), a code fragment library (110), and development standards in order to create modular code, thereby addressing limitations of the prior art.



FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AL	Albania	ES	Spain	LS	Lesotho	SI	Slovenia
AM	Armenia	FI	Finland	LT	Lithuania	SK	Slovakia
AT	Austria	FR	France	LU	Luxembourg	SN	Senegal
AU	Australia	GA	Gabon	LV	Latvia	SZ	Swaziland
AZ	Azerbaijan	GB	United Kingdom	MC	Monaco	TD	Chad
BA	Bosnia and Herzegovina	GE	Georgia	MD	Republic of Moldova	TG	Togo
BB	Barbados	GH	Ghana	MG	Madagascar	TJ	Tajikistan
BE	Belgium	GN	Guinea	MK	The former Yugoslav Republic of Macedonia	TM	Turkmenistan
BF	Burkina Faso	GR	Greece			TR	Turkey
BG	Bulgaria	HU	Hungary	ML	Mali	TT	Trinidad and Tobago
BJ	Benin	IE	Ireland	MN	Mongolia	UA	Ukraine
BR	Brazil	IL	Israel	MR	Mauritania	UG	Uganda
BY	Belarus	IS	Iceland	MW	Malawi	US	United States of America
CA	Canada	IT	Italy	MX	Mexico	UZ	Uzbekistan
CF	Central African Republic	JP	Japan	NE	Niger	VN	Viet Nam
CG	Congo	KE	Kenya	NL	Netherlands	YU	Yugoslavia
CH	Switzerland	KG	Kyrgyzstan	NO	Norway	ZW	Zimbabwe
CI	Côte d'Ivoire	KP	Democratic People's Republic of Korea	NZ	New Zealand		
CM	Cameroon			PL	Poland		
CN	China	KR	Republic of Korea	PT	Portugal		
CU	Cuba	KZ	Kazakstan	RO	Romania		
CZ	Czech Republic	LC	Saint Lucia	RU	Russian Federation		
DE	Germany	LI	Liechtenstein	SD	Sudan		
DK	Denmark	LK	Sri Lanka	SE	Sweden		
EE	Estonia	LR	Liberia	SG	Singapore		

METHODS AND APPARATUS FOR NETWORK APPLICATIONS USING OBJECT TOOLS

CROSS-REFERENCES TO RELATED APPLICATIONS

This application claims the benefit of U.S. Provisional Application No. 60/079,611, filed March 27, 1998, hereby incorporated by reference.

TECHNICAL FIELD

The present invention relates, generally, to network groupware applications and, more particularly, to methods and apparatus for developing object-oriented systems in the context of non-object-oriented application environments.

BACKGROUND ART AND TECHNICAL PROBLEMS

Object oriented programming is a formal programming practice characterized by, among other things, encapsulation, inheritance, and object polymorphism. Due in part to the advantages of these characteristics, object oriented programming has quickly become the preferred software development paradigm. Notwithstanding this trend, modern enterprise groupware database systems remain tied, on the development level, to traditional programming techniques. Such systems, while powerful, are substantially aimed at providing ease of use for end-users, and therefore rely almost exclusively on simple graphical programming methods and flat-file functionality. Example enterprise groupware environments include, for example, the Lotus Notes application environment as described in the relevant system documentation, e.g., LOTUS NOTES RELEASE 4 APPLICATION DEVELOPER'S GUIDE (1995), hereby incorporated by reference.

The lack of modular and object oriented design standards in known groupware database applications gives rise to a number of problems. For example, application development time is often unnecessarily prolonged due to a lack of code reuse. That is, programming efficiency is decreased when custom modules must be created for a given application. In addition, it is often difficult to customize a particular section of code when such customization is desired. That is, because of the inherent non-

modularity of typical enterprise groupware environments, altering one segment of code can unexpectedly impact on other sections of code in undesirable ways.

Furthermore, programmers are not always disciplined with respect to naming standards for forms, views, fields, variables, and the like. As a result, other programmers (indeed, even the same programmer) may be confronted by names which are inconsistent and fail to describe the role of the particular component. This tends to increase development time, generate bugs, and frustrate code maintenance. Moreover, such inconsistencies tend to reduce code-sharing as it is often difficult for one programmer to understand, adapt, and maintain the code written by another.

When clear software standards are not employed, quality control is very difficult to implement. Moreover, such coding tends to be inconsistent across application, resulting in variation in user interfaces experienced by the end user. Even when clear standards exist, it is extremely difficult and time-consuming to manually search through code to locate inconsistencies and bugs.

Methods and apparatus are therefore needed to overcome these and other limitations in the prior art.

SUMMARY OF THE INVENTION

The present invention provides methods and apparatus for developing object-oriented network groupware applications in substantially non object-oriented programming environments. Object-oriented architectural and naming standards are used in conjunction with object repositories, an application builder, an application analyzer, a code fragment library, and development standards in order to create modular code, thereby addressing limitations of the prior art.

In accordance with one aspect of the present invention, a method for using a programmable digital computer to create object-oriented applications within a non-object-oriented software environment includes the steps of: creating, in accordance with a predetermined set of design standards, an object within said non object-oriented software environment, wherein: said object comprises at least one of said design elements; said object is characterized by inbound public interfaces, outbound interfaces, and dependencies implemented using said design elements; transferring

said object to said application; and determining, for said object, a level of compliance to said predetermined set of design standards.

BRIEF DESCRIPTION OF THE DRAWING FIGURES

The subject invention will hereinafter be described in conjunction with the appended drawing figures, wherein like numerals denote like elements, and:

Figure 1 shows a diagrammatic block diagram of an exemplary object tools system;

Figure 2 shows a conceptual model of an exemplary object useful in illustrating the present invention;

Figure 3 is a flowchart depicting an exemplary application development process;

Figure 4 shows a diagrammatic block diagram of an exemplary network environment;

Figure 5 shows an exemplary application builder main menu;

Figure 6 shows an exemplary menu for use in an "add objects" function;

Figure 7 shows an exemplary menu for use with a "remove objects" function;

Figure 8 shows an exemplary menu for use with an "import new objects" function;

Figure 9 shows an exemplary menu used to invoke an Application Analyzer in accordance with the present invention;

Figure 10 presents a general guide to object model diagrams; and

Figure 11 shows an object model diagram corresponding to an exemplary architectural standard.

DETAILED DESCRIPTION OF PREFERRED EXEMPLARY EMBODIMENTS

A system in accordance with various aspects of the present invention includes methods and apparatus for developing object-oriented network groupware applications in substantially non object-oriented programming environments. With reference to **Figure 1**, an exemplary object tools system 100 includes Carrier Databases 102, Object Repositories 104, Application Builder 106, Code Fragment Library 110, Application Analyzer 114, and Analysis Template 112. All or some of

these components may be employed in creating the individual software applications 108. In addition, an exemplary embodiment might include various tutorial and reference materials, e.g., user guide 118, example databases 120, and walk-through databases 122. An auxiliary file database may also be provided for storing ancillary files used by various objects, for example, graphic files, Java applets, and the like. It will be appreciated that **Figure 1** presents a conceptual, diagrammatic overview of an exemplary object tools system in accordance with the present invention, and that suitable hardware and software components (e.g., servers, routers, data links, user computers, and the like) would be employed in order to implement the present system in a network environment. The specific geometry, topology, and labels used in **Figure 1** are not intended to limit the scope of the present invention.

Referring now to **Figure 4**, the present system is suitably implemented in a distributed enterprise environment comprising one or more servers 402, one or more developer workstations 405, and one or more user workstations 404 connected using appropriate data links 410 over a conventional network 408. The various objects and applications might be located within servers 402, user workstations 404, or a combination of both. Software development preferably takes place on developer workstations 406, but may alternatively take place at any other suitably configured workstation (e.g., servers 402 or user workstations 404). Servers 402 and user workstations 404 may comprise any suitable hardware/software configuration, for example, any of the various Intel Pentium-based computers or equivalents running in a Windows NT or Windows 95 environment , or various Sun Microsystem workstations running in a Solaris environment. In a preferred embodiment, developer workstations comprise a 32 bit Windows NT or Windows 95 workstation.

Referring again to **Figure 1**, Carrier Databases 102 provide a mechanism through which the various objects may be distributed to end-users via e-mail, the Internet, or any other conventional data communication network. Carrier Databases 102 allow objects to be shared between users and provide a efficient means for developers to distribute updated objects to users utilizing those objects in their applications.

Object Repositories 104 are preferably used to store the various design elements that comprise the objects used to build individual applications 108. User's may create their own Object Repositories 104, and common repositories may be provided, for example, as a location for archiving old and outdated objects.

Application Builder 106 assists the developer in transferring objects within the system, and acts as an interface to Application Analyzer 114 and various standards and preferences related thereto. In an exemplary embodiment, Application Builder 106 allows transfer of objects to and from Object Repositories 104, applications 108, and Carrier Databases 102. This allows the developer to easily add or remove objects from an application in a robust manner -- i.e., in a way that is not likely to generate unexpected software bugs.

Code Fragment Library 110 comprises small pieces of reusable code -- preferably designed in accordance with standards set forth in greater detail below -- which are typically not themselves complete functions or subroutines, but which can easily be copied into objects during application development.

Application Analyzer 114 is preferably used to analyze individual or groups of applications 108 as well as the various objects that comprise those applications. In a preferred embodiment, Application Analyzer 114 creates a Result Document 116 for each design element and object used in an application 108 and sorts these result Documents 116 into predefined categories for further analysis. Application Analyzer 114 may also be used to identify any deviations from the preferred architectural and naming standards, thereby promoting standards compliance. In this way, Application Analyzer 114 facilitates efficient quality control, debugging, and preparation of documentation for applications 108. Documents 116 are preferably created in accordance with Analysis Template 112, which specifies a format for the various reports included in Result Documents 116.

Objects

Before discussing the various system components in detail, it is necessary to define the term "object" as used in the context of the present invention. As noted above, it is common for database environments -- for example, Lotus Notes -- to

provide user interfaces which are end-user-friendly, but which are not object-oriented, and therefore suffer from a number of development limitations as described above in the Background section. The present system utilizes the underlying elements of the database environment, but imposes upon them and profits from an object-oriented framework.

Object-oriented frameworks offer a number of advantages.

Having thus given a general overview of object-oriented programming, a method for employing object-oriented techniques in the context of the present invention will now be described. In the discussion that follows, the terminology popularized by Lotus Notes will be used throughout. Those skilled in the art, however, will appreciate that the present invention may be deployed in other database environments. For example, the meaning of "object" should not be construed to include only conventional Lotus Notes elements. Other object types (Dynamic HTML, APIs, query layouts, etc.) may also be employed.

Referring now to **Figure 2**, in accordance with one embodiment of the present invention, an object may comprise any combination of forms 202, subforms 208, views 204, agents 206, navigators 210, and fields 212. An object may also comprise groups, roles, script subroutines or class definitions, graphics, or portable code files (not shown in **Figure 2**).

Briefly, a form is a template, or window, through which the user may view database information. Forms, which are roughly analogous screen layouts, present data in a number of fields in a manner specified by the application developer.

A subform is a type of form containing fields configured in an often-used layout (for example, header information, navigation bars, and the like).

A view is used to display database information in row and column format in order to assist the user in finding the desired information. A view may include data extracted from fields, results, or computation fields. A navigator may also be used to provide graphical assistance in finding data.

A field contains a particular type of information, for example, text, rich-text, word lists, numbers, times, dates, user names, etc. In a typical application, a form presented to the user includes a number of fields, some of which are configured to

receive user input, and some of which are constant or a function of other fields or data. In addition, appropriate security conditions may be associated with a given field, sets of fields, documents, databases, applications, or data records.

An agent is a component that automates certain functions within an application. For example, an agent may be employed to notify users of an impending due date, perform database maintenance, or perform various types data manipulation.

Further information regarding these and other exemplary object elements may be found in a number of texts, for example, the Lotus Notes 4 Application Developer Guide referenced above.

Given the exemplary set of object elements described above, the manner in which these elements may be employed in an object oriented environment will now be outlined.

An important aspect of object-oriented systems is the distinction between private and public elements. An object is, by definition, functionally abstract -- meaning that while its functions and public interfaces are quite clear in usage, its internal workings are not necessarily clear or even known to the user. In the present invention, design elements (forms, views, etc.) may be designated as public or private, and fields themselves may also be public or private. That is, certain forms, views, subforms, agents, and navigators may be used by the object itself, or may be used by other objects as well. Similarly, certain fields may be public (referred to as "out-bound public interfaces") or private (in which case they must not be referred to by any other object). For example, in **Figure 2**, field 212(a) is illustrated as public, while field 212(b) is illustrated as private.

In the context of the present invention, objects receive input through fields, referred to as "in-bound public interfaces." These fields are created and maintained in the referring code, not the object itself. For example, field 212(a) shown in **Figure 2** may be an out-bound public interface to another object which uses the value of field 212(a) as an in-bound public interface to an internal algorithm.

Objects may also receive inputs from documents, i.e., they may retrieve input from data stored elsewhere in the database.

In accordance with a the present methodology, an object should not be thought of as simply a field, or an agent, or as any particular element. Instead, an object is a function or component whose behavior and implementation is defined as a collection of design elements and properties. This provides a mechanism which makes it easy to define complex objects that are not pure instances of one element. A particular object might exist in only one part of the environment, or it might extend into several parts of the environment. For example, a basic field object might only exist in the field and form parts of the environment, whereas a object free-time search algorithm would probably involve fields, views, agents, buttons, and layout regions.

Interfaces are essentially any location that you can put code in the underlying groupware application, for example, events, view selection formulas, agents, script subroutines, navigators, icons, properties, screen "hot-spots," and the like.

Development Process

Having thus given a general overview of objects as well as various components comprising an exemplary object tools system 100, an overview of an exemplary development process in accordance with various aspects of the present invention will now be described.

Referring now to **Figure 3**, a developer typically begins a design session in one of three ways. First, a developer may utilize a Carrier Database 102 to share preexisting objects with another Developer (Step 304). Alternatively, the developer may proceed directly to invoking Application Builder 106 (Step 310), or may modify or debug a pre-existing object (Step 308).

In Step 304, one or more Carrier Databases 102 are preferably utilized as a vehicle for assembling new or modified objects prior to inclusion in an Object Repository 104. Carrier Databases 102 are preferably created from a carrier database template residing in a suitable location within the system. The developer preferably populates the Object Repository 104 with at least the following items: 1) design elements used by the object, 2) documentation for the object, and 3) an object map document (listing object names, object IDs, design elements, interfaces, and the like).

In Step 308, the developer assembles and configures the various forms, subforms, views, agents, navigators, and fields that are required for a particular application. These design elements are preferably created, named, and configured in accordance with architectural and naming standards described in detail below.

In Step 310, the developer invokes Application Builder 106. As mentioned briefly above, Application Builder 106 supports modularity by assisting the developer in transferring objects in a robust manner. In addition, Application Builder 106 acts as an interface to Application Analyzer 114 and Code Fragment Library 110 and can be used to define other interfaces created by the developer. In an exemplary embodiment, Application Builder 106 allows transfer of objects to and from Object Repositories 104, applications 108, and Carrier Databases 102. Specifically, when the developer initiates Application Builder 106, the system presents a suitable graphical interface for selecting from a plurality of functions. In an exemplary embodiment, a main menu screen such as that shown in **Figure 5** is displayed, offering at least four choices: add objects 502, remove objects 504, import objects 506, and run Application Analyzer 508. If the developer selects the "add objects" function (502), a second, more detailed input screen is displayed, for example, an input screen as shown in **Figure 6**. Here, the developer selects the appropriate application to which the objects will be added (602), selects the objects to be added (604), and implements the addition with certain options (606). In this way, the developer is able to transfer objects from various Object Repositories to the application of interest (items 104 and 108 in **Figure 1** respectively).

Referring again to **Figure 5**, "remove objects" function 504 allows the developer to remove unwanted or unneeded objects from an application. A variety of interfaces to this function may be appropriate – for example, an interface as shown in **Figure 7**. More particularly, in the illustrated embodiment, one or more menu screens suitably provide an input region 702 for selecting an application from which objects are to be removed, an input region 704 for selecting objects to removed, a region 706 for activating the remove function, and an input region 708 for reviewing a log of past removal operations.

"Import objects" function 506 suitably allows the developer to move an objects design elements, documentation, and object map from a Carrier Database 102 to an Object Repository 104. A variety of user interfaces may be employed to carry out this function. In an illustrated embodiment, one or more menu screens provide a region 802 for selecting an Object Carrier from which objects will be imported, a region 804 for selecting a Repository to which objects will be transferred, a region 806 for specifying the location of the applicable user guide (i.e., documentation such as object models, required elements, and the like), a region 808 for choosing object types (i.e., subsets of available objects), a region 810 for activating the import function, and an input region 812 for reviewing the log.

Application Builder 108 also suitably provides an option for accessing the Code Fragment Library 110 (function 510). It will be appreciated that the various functions presented by the illustrated Application Builder 106 may alternatively be distributed between multiple software programs using any number of different user interfaces.

In cases where objects are added to an application, the developer may be required -- in accordance with certain architectural standards as detailed below -- to modify or add objects, fields, forms, or other design elements. Such required modifications are specified in the documentation via an object map associated with a given object. More particularly, documentation associated with each existing object preferably sets forth a list of objects comprising that object, and specifies the required and optional inputs for each form. These inputs, or inbound public interfaces, are then supplied by the developer within the object or form being created. The documentation further specifies any dependent objects which must be copied (using Application Builder 106) and suitably configured.

"Run application analyzer" function 508 suitably invokes the Application. (Step 312). As mentioned briefly above, Application Analyzer 114 is preferably used to analyze individual or groups of applications 108 as well as the various objects that comprise those applications. In a preferred embodiment, Application Analyzer 114 creates a result document 116 for each design element and object used in an application 108 and sorts these documents 116 into predefined categories for further analysis. Application Analyzer 114 is also used to identify any deviations from the

preferred architectural and naming standards set forth below, thereby promoting standards compliance. In addition, Application Analyzer 114 may be used to incorporate various standards developed within an end-user's organization, assess an application's Year-2000 compliance, and/or search for non-applicable or obsolete functions or code.

In an exemplary embodiment, Application Analyzer 114 is invoked through Application Builder 106, for example, as shown in **Figure 5**, where function 508 provides an entry point into the analyzer. Running Application Analyzer 114 (Step 312) involves selecting an application to analyze (through selection of the appropriate server and file path name), selecting an analysis file to which the results will be posted, then selecting various options as desired (for example, choosing whether incremental analysis should be performed and whether the objects themselves should be checked). More particularly, an illustrated embodiment employs one or more menu screens as shown in **Figure 9** wherein the user/developer is presented with an input region 902 for selecting the application, input region 904 for selecting an analysis file, and a region 906 for invoking the analyze function.

When the analysis is complete, Application Analyzer 114 preferably creates a Result Document 116 for developer review which lists various data helpful in debugging and documenting the application in question (Step 314). Specifically, Application Analyzer 114 preferably lists the various inspected objects along with any naming or architectural standards that have been violated.

After reviewing the results of the Application Analyzer, the developer may choose to debug the application under development to reconcile any deviations from the architectural or naming standards (Step 316). Thus, the developer may choose to return to the Step 302 (or, most likely, Step 306) to continue refining and developing the application or various objects within the application.

Standards

As mentioned above, one aspect of the present invention relates to an advantageous set of standards imposed on object development which are "enforced" via Application Analyzer 114. These standards are aimed primarily at producing

readable code, facilitating easy debugging, encouraging code reuse, and enhancing modularity. In general, standards used in the context of the present invention can be categorized as either naming standards or architecture standards.

Naming standards involve rules used in conjunction with design elements (forms, views, etc.) which are intended to incorporate into the element's name some indication of the purpose and/or source of the element. Architecture standards relate to, among other things, framework or "kernel" objects and design elements which are required in all or most applications. In addition to naming standards and architectural standards, certain visual standards related to text and graphics layouts may also be desirable.

With respect to naming standards, each object is preferably named in such a way as to affect the various goals of the present invention. In a preferred embodiment, the object name is constructed as:

<element prefix><object ID>-<element name>, or

<element prefix><object ID><instance number><element name>,

wherein the size and description of each component is specified in **Table 1** below.

Component	Number of Characters	Description
<element prefix>	Two or Three	Describes what the element is and how it is used. Most types of design elements have several possible prefixes.
<object ID>	Six (with leading zeros and trailing x's if needed)	A unique ID that is assigned to the object for identification purposes. All design elements that are part of the same object have the same object ID in their names. This enables programmers and automated tools to easily determine the object to which a design element belongs.
Hyphen or <instance number>	One	A hyphen is used mainly for visual distinction between the prefix and the descriptive part of the name, making it easier to read a list of design element names. An "instance number" is used when multiple copies of an element in one database are needed. For example, there may be a need for two subforms which are substantially identical and serve the same purpose but must have different names so that they may be used together on the same form. The instance number is preferably a single digit, 0 through 9. If more than 10 instance numbers are needed, lowercase alphabetic characters should be used.
<element name>	Variable	Descriptive name for the design element which identifies what the element is used for. Mixed case with the first letter of each distinct word capitalized. May include valid punctuation characters, but no spaces, avoiding certain symbols and punctuation characters (e.g., " \","#", "@", and "*") which may cause problems for Web clients.

TABLE 1

Object prefixes are preferably specified for each class and type of design elements. In the case of view elements, prefixes are suitably chosen in accordance with **Table 2** below.

Prefix	Comment
cal	A calendar view, visible to the user.
db	A view used internally by the application for lookups or other processing. Not visible to the user.
fol	A folder, visible to the user.
vdb	A view used internally and also seen by users (such as views used in pick-lists).
vv	Views visible to the user. Not used for any other purpose, such as lookups, iterating through a set of documents, etc.

TABLE 2

For example, the name "dbAC0012-PartNumberLookup" may be used in conjunction an internal look-up table view specifying a list of part numbers.

It may be desirable to create separate hidden views for users and for any internal processing views performed by an application, making it more convenient to alter the design of elements apparent to the user without affecting the code, thereby reducing the possibility of introducing errors. The "vv"/"cal"/"fol" and "db" prefixes are designed to make this distinction clear.

For visible views (i.e., views which appear on the View menu or the View/Folders pane) the IOT name is preferably used as the view's alias, as in "All Parts\By Size | vvAC0012-AllPartsBySize".

In a preferred embodiment, element prefixes for forms are selected in accordance with **Table 3** below.

Prefix	Comment
doc	Regular document form
res	Response form
rtr	Response-to-response form
dlg	Document form used for a dialog box
ntp nav (obsolete)	Form used for a navigator template
vtp	Form used for a view template
vsr	Form used for a Search Results Template

TABLE 3

For forms whose names will not appear on the "Create" menu, object names with no aliases are preferably used (e.g., "res000000-CustomerComment"). For forms whose name will appear on the "Create" menu, the format "Create Customer Comment | res000000-CustomerComment" is preferably used. If in the context of a specific enterprise software system an alias is required (as might be the case with Notes or Domino), the alias is preferably placed to the right of the object name, for example: "ntpAC0012-StandardWebUI | $$NavigatorTemplateDefault."

In a preferred embodiment, element prefixes for subforms are selected in accordance with **Table 4** below.

Prefix	Comment
sub	Regular subform

TABLE 4

For subforms, aliases are not needed, and only the object name need be used (for example: "sub0034xx-StandardDocumentHeader").

Script Libraries preferably employ a single element prefix as specified in **Table 5** below.

Prefix	Comment
scl	A script library

TABLE 5

As with subforms, script libraries do not require aliases. For script libraries that contain a single class definition (with or without derived classes), the convention of naming the library with "class" plus the class name is preferably used.

Navigators preferably use a single element prefix as specified in **Table 6** below.

Prefix	Comment
nav	A navigator

TABLE 6

There are preferably two types of navigator naming schemes -- one for navigators that will appear on the View\Show menu, and one that will not. For navigators whose names will appear on the View\Show menu, an alias is preferably used, for example, "End User Menu | navAC8020-EndUserMenu". For navigator names that will not appear on the View\Show menu, the object name within parenthesis is preferably used, as in "(navAC0820-ConfigMenu)".

In an illustrated embodiment, a more complex prefix scheme is used for agents since each one of the three characters in an agent prefix has a distinct meaning. More particularly, the first character identifies how the agent is triggered; the second character identifies how often the agent should run if it is a scheduled agent; and the third character identifies the documents to be processed (i.e., the "run on" setting).

Preferred characters for the first, second, and third places of the agent prefix are specified in **Tables 7, 8, and 9** respectively.

First Prefix Character	How the Agent is Triggered
s	Scheduled background agent
t	Action menu or @Command([ToolsRunMacro]..)
a	Called by another agent
m	When documents are mailed to the database/new mail
p	When documents are pasted
v	Domino QuerySave agent
o	Domino QueryOpen agent

TABLE 7

Second Prefix Character	How the Agent is Scheduled
x	Placeholder if the agent is not a scheduled agent
h	Scheduled agent runs hourly
d	Scheduled agent runs daily
w	Scheduled agent runs weekly
m	Scheduled agent runs monthly
0	Scheduled agent runs every half hour
1	Scheduled agent runs every hour
2	Scheduled agent runs every two hours
4	Scheduled agent runs every four hours
8	Scheduled agent runs every eight hours

TABLE 8

Third Prefix Character	Type of Document Agent Runs On
a	All documents in the database
m	Runs on newly mailed documents
n	New and modified documents (i.e., the agent's unread marks)
p	Runs on pasted documents
u	Unread documents (i.e., the user's unread marks)
v	All documents in the view
s	Selected documents
r	Run once

TABLE 9

If the agent is specified to run at a fixed frequency, the digits 0-8 are preferably used for the second character. However, if the agent can be set by the user to run at a different frequency, the letter "h" ("hourly") is preferably used. For scheduled agents, the letters preferably "a" and "n" are used for the third character of the prefix.

Agent aliases can be complex, particularly in the context of Notes, since Notes itself will modify the agent name when the agent is set to "Run From Agent List." For agents whose names will appear on the Actions menu, an alias format such as "Process Sales Leads | txsAC8020-ExportLeads" is preferably used. For agents whose names are to appear on the View\Show menu, the object name is preferably used within parenthesis, for example: "(txrAC0820-ConfirmationPrompt)".

An alternate way to hide an agent is to set its run option to "Manually From Agent List." Notes will modify the names of such agents to include the parenthesis automatically. Aliases should not be used when naming these agents, as they are not necessary: the IOT name alone is preferably used.

In the illustrated embodiment, Object IDs are specified as comprising exactly six characters, wherein a character may consist of a letter (upper case or lower case) and some symbols. For readability purposes, it is advantageous to limit characters to upper case letters and numbers, using lower-case "x"s as place-holders where necessary (for example, "0123xx", "TESTxx", etc.) In this regard, **Table 10** sets forth a preferred object ID naming scheme, where "?" denotes any alphanumeric

character.

ID	Description
000000	An all-zero object ID indicates a design element that is not associated with a reusable object. These are most commonly used for elements that are specific to an application.
0001?? through 9999??	Object IDs that begin with four numeric characters may be reserved for use by the development software and other products
A????? through Z?????	Object IDs in this range are available for developers to assign to objects in custom repositories. For example, if the developer's corporate initials are "AC" the developer might assign IDs in the sequence "AC0001" through "AC9999" to be custom objects.
TEMP?? DEMO?? TEST??	Object IDs that begin with these characters are used for elements that are not intended to become part of a finished application or object. The Application Analyzer preferably warns the developer when it comes across such an ID.

TABLE 10

A Notes field name is constructed as:

<purpose><list><datatype prefix>_{object ID}<field name>, or

<purpose><list><datatype prefix><instance number>{object ID}<field name>.

Table 11 below sets forth a description of each of these components in accordance with a preferred embodiment of the present invention.

Component	Number of characters	Description
<purpose>	Zero or One	Indicates whether the field is used in any special context, such as a local variable. There are only a few cases where the purpose character is used.
<list>	Zero or One	A lower case "l" for fields that can contain multiple values.
<datatype prefix>	One to Four	Indicates the data type stored in the field. This tag covers all the Notes field data types (Text, Number, Rich Text, etc.) and also includes some additional ones where more detail is needed.
Underscore or <instance number>	One	The hyphen is used mainly for visual distinction between the prefix and the descriptive part of the name, making it easier to read field names. An instance number is used when there is a need to have multiple copies of a field in a database. For example, there may be a need for two computed-for-display fields to display the same information in multiple places. One might be named "dtxt1PhoneNumber," the other "dtxt2PhoneNumber," making it clear that they are intended to show the same information. The instance number must be a single digit, 0 through 9. If more than 10 instance number are required, lowercase alphabetic characters may be used.
{object ID}	Zero or Six	Optional component: used only when there is a chance that a field name in one object will duplicate one in another object. Note that if descriptive field names are chosen, the chances of name-collision are minimal.
<field name>	Variable, up to the limit for field names	Indicates the purpose or content of the field. A descriptive, self-documenting name is preferably used. The convention is to use mixed-case, and to capitalize the initial letter of each word. Spaces, punctuation, and most symbols are preferably not used.

TABLE 11

It should be appreciated that, for CGI surrogate variables (i.e., variables using the prefix "HTTP_"), capital letters should be used in order to duplicate the precise name

of the CGI variable. Examples field names in accordance with the above scheme include: "txt_OriginalAuthor" for a regular text field; "num_ACME54CurrentPrice" for a numeric field belonging to Acme object 54; and "dlnam_StateApprovers" for a multi-value, computed-for-display name field.

As mentioned briefly above, the first component for a field name is the Purpose identifier. In a preferred embodiment, purpose identifiers conform to the standards set forth in **Table 12** below.

Purpose Identifier	Comment
d	Field is computed-for-display
v	Local variable that is used internally in a formula, and does not correspond to a field
k	Key field (not to be confused with a keyword field) which uniquely identifies a document
f	Foreign key field (another document's key field value)
<none>	None of the above

TABLE 12

Example Purpose identifiers in accordance with this system include: "dtxt_PhoneNumber" for a computed-for-display field; "vnam_CurrentAuthor := nam_DocAuthor" for a local variable in an @Formula being set to the value of a field on the current document; and "ktxt_EmployeeSSN" for a field containing a unique identifier for the document.

The second component for a field name is the Data Type prefix. Data Type prefixes preferably conform to the scheme set forth in **Table 13** below.

Data Type Identifier	Comment
txt	Text field
tim	Date/time field
num	Number field
sec	Section field
aut	Author Names field
rdr	Reader Names field
nam	Names field
rtf	Rich-text field
html	Text field containing an HTML string (Hypertext Mark-up Language)
HTTP HTTPS	Text field corresponding to a CGI variable (Common Gateway Interface)
url	Field that contains a World Wide Web URL (Uniform Resource Locator)
yn	Field that has either a "Yes" or a "No" value
f	Numeric field that has a Boolean value of either @True or @False. "f" stands for "flag."
unid	Text field that contains a Notes Document Unique ID (@DocumentUniqueID)
list (obsolete)	A multi-value text field. The current standard uses a lowercase "l" before the datatype prefix to indicate multi-value fields instead of this obsolete prefix.
nid	Text field that contains a Note ID (@NoteID)

TABLE 13

Example Data Type prefixes in accordance with these rules include: "num_EditCounter", "tim_Created", "html_AppletTag", "rtf_Body", and "HTTP_HOSTNAME".

Object Identifier fields may be used to detect the presence of an object within a document. For example, if an object designated as "IIO0033 Expiration" is used on a form, documents created with that form will contain a field named "IIO0033." The Expiration agent, or any other object in the application, can then detect such documents using a selection formula such as "SELECT @IsAvailable(IIO0033)".

Object Identifier fields preferably consist of an object ID and an optional descriptive label in the format:

<object ID>

or,

<object ID>_<label>

In a preferred embodiment, Object Identifier fields are always text fields, and are usually computed-when-composed. Any object that includes either a form or a subform should include an object identifier field, typically placed at the top of the form or subform. Since complex objects may consist of several types of documents, an object may have more than one object identifier field.

With respect to Script variables, there are generally two types: (1) Variables of fundamental data types, such as Integer, Double, String, Lists, Arrays and user-defined types; and (2)Variables that refer to an instance of a class, i.e., an object variable declared to be a NotesDocument, NotesDatabase, or some other class. Names for fundamental data type scripts are preferably constructed as follows:

<scope/parameter type><aggregate type><data type prefix>_<variable name>{suffix} ,

while names for instance variables are constructed as:

<scope/parameter type><aggregate type><class prefix>_<variable name>{suffix} ,

or, in an abbreviated form:

<scope/parameter type><aggregate type><class prefix>{_suffix}.

Script components preferably conform to the guidelines set forth in Table 14.

Component	Number of Characters	Description
<scope/parameter type>	Zero or One	"Scope" is the visibility of the variable in a function/procedure. For example, variables may be defined locally or globally. Using a prefix for scope is especially important in LotusScript where the global module is well hidden, as one might wonder where specific variables have been declared. "<parameter type>" refers to how a parameter appears in a function or subroutine; either "by reference" or "by value." The distinction is important because changes to a "by reference" parameter can have inadvertent side-effects beyond the scope of a given function.
<aggregate type>	Zero or One	Used to identify variables of the built-in "Aggregate" data types: arrays and lists.
<data type prefix>	One to Four	Indicates the fundamental type of the variable. In addition to the basic data types like String and Integer, the system may define some extra prefixes for convenient types. Preferably never used in conjunction with a <class prefix>.
<class prefix>	Three to Six	Indicates the class to which an object variable belongs. Some user-defined classes may specify a <class prefix> to use for instances of that class, but it is not required. If no class prefix is defined for objects of a given class, the "obj" prefix should be used instead. Preferably never used in conjunction with a <data type prefix>.
Underscore	One	For readability, the underscore character comes between the data type or class prefix and the descriptive name.
<variable name>	Variable, up to the limit for Script variable names	The descriptive name of the variable. Preferably use mixed case with initial capitals on distinct words. The variable name can be omitted if it is clear that there is only one possible instance of a variable of that type.
{suffix}	Variable	Optional, but quite convenient to use for naming variables.

TABLE 14

Scope/Parameter Types preferably conform to the guidelines set forth in **Tables 15 and 16** below.

Scope	Comment
g	Global variable that has been declared in a "Globals" section
m	Module level variable that has been declared in the "Declarations" section of a form, button, agent, etc.
<none>	Local non-parameter variable declared in function/procedure

TABLE 15

Parameter Types	Comment
r	Variable is a reference parameter in a function/procedure. Reference parameters return their values to the calling function/procedure. This is the default in LotusScript.
v	A "ByVal" parameter to a function/procedure. Value parameters do not return their values to the calling function/procedure.

TABLE 16

Aggregate Types preferably conform to the guidelines set forth in **Table 17** below.

Special Type	Comment
a	Variable is an array
l	Variable is a list (**NOTE:** Lotus Script lists are different from Notes multi-value list fields, though both use the lower case "L" character in their names.)
<none>	Variable is a single variable

TABLE 17

Data Type prefixes, Object Class prefixes, and Suffix prefixes preferably conform to guidelines set forth in **Tables 18, 19, and 20** below.

Data Type	Comment
dbl	Double precision floating point number
sng	Single precision floating point number
cur	Currency
lng	Long integer
int	Integer
f	An Integer variable containing either True or False. "f" stands for "flag variable"
var	Variant
dat	Date/Time
str	String
typ	A variable whose type is a user-defined data type.

TABLE 18

Object Class Prefixes	Comment
obj	Variables that are instances of user-defined classes, or of classes that do not have an associated class prefix. For example, "obj_AnalysisDoc" would indicate an instance of the user-defined class "class_AnalysisDoc".
nacl	NotesACL
nacle	NotesACL Entry
nagt	NotesAgent *
ndb	NotesDatabase *
ndt	NotesDateTime
ndir	NotesDbDirectory
ndoc	NotesDocument
ndtr	NotesDateRange
ncol	NotesDocumentCollection
neo	NotesEmbeddedObject
nfrm	NotesForm *
nintl	NotesInternational
nitm	NotesItem *
nlog	NotesLog
nnam	NotesName
nnews	NotesNewsletter
nreg	NotesRegistration
nrsty	NotesRichTextStyle
nrti	NotesRichTextItem *
nses	NotesSession
ntmr	NotesTimer
nuidb	NotesUIDatabase
nuidoc	NotesUIDocument
nuivw	NotesUIView
nuiwsp	NotesUIWorkspace
nvw	NotesView (It is preferable to name objects that have corresponding Notes element or field names by using the same descriptive name both in the Notes database and in Script).
nvcol	NotesViewColumn
oconn	ODBCConnection
oqry	ODBCQuery
ores	ODBCResultSet

TABLE 19

Data Type	Comment
Cur	Current element (e.g. of array, result set, list etc.)
First	First element
Last	Last element
Next	Next element
Prev	Previous element
Min	Lower limit of range (e.g. of an array)
Max	Upper limit of range
Src	Source (e.g. of copy operation)
Dest	Destination
Old	Old element
New	New element
Temp	Temporary element

TABLE 20

Example script variable names in accordance with these guidelines include: "gdat_Actual" for a global date/time variable; "ndoc_Cur" for a "current" document in a loop; "lstr_CustomerIDs" for a list of strings; and "gatyp_MyArray( )" for a global array of a user-defined type.

A LotusScript constant name is preferably constructed as:

<scope><data type prefix>_<CONSTANT NAME>

wherein the name components conform to the standards set forth in Table 21 below.

Component	Description
<scope>	"g" for global, "m" for module, or omitted for local constants
<data type prefix>	One of the data type prefixes defined for Script variables.
<CONSTANT NAME>	Description of the constant name. Use all capital letters, with distinct words separated by underscores.

TABLE 21

Example constant names include: "gnum_HOLIDAYS_PER_YEAR" for a global numeric constant; and "str_JANUARY" for a string constant named "January". Similarly, user defined type names are preferably constructed as:

type_<type name> , and

class names are preferably constructed as:

class_<class name> .

Functions are also preferably given data type prefixes to reflect the type of value the function returns, i.e.:

<data type prefix>_<function name> ,

wherein the data type prefixes for function names are identical to those for script variables set forth above (for example: Function int_DaysBetweenDates(rdat_First as Variant, rdat_Last as Variant) As Integer).

Architectural Standards

As mentioned briefly above, a system in accordance with the present invention provides and enforces a set of advantageous architectural standards which are imposed on the development process, thereby affecting the larger goal of providing object-oriented functionality in the context of a non-object-oriented software environment. Architectural standards relate to, among other things, framework or "kernel" objects and design elements which are required in all or most applications.

Architectural standards are preferably documented and communicated graphically via an object modeling technique (OMT). In the present context, an OMT (for example, the Unified Model Language, or UML) is a technique for representing design elements and the relationships between those elements in a concise graphical format. The OMT diagrams can be though of as "blueprints" for the objects.

Referring now to **Figure 10**, OMT notation involves the use of bi-level boxes to represent design elements, wherein the top box (e.g., 1001) includes the name of the element, and the bottom box (e.g., 1003) sets forth various attributes of the element, if appropriate. Boxes that do not include a design element name are referred to as "abstract," and represent either other objects, groups of elements, interfaces, or non-implemented (virtual) base classes. Relationships between objects are

designated by lines (e.g., 1005), which may or may not have an associated descriptive label. The nature of the relationship between two elements is denoted by the use of particular symbols at one or more endpoints of the line. More particularly, with continued reference to the graphical conventions shown in **Figure 10**, a one-to-one association (1002) indicates that element A is associated with exactly one B; a zero-or-more association (1004) indicates that A is associated with zero or more Bs; a one-or-more association (1006) indicates that A is associated with one or more Bs; a Zero-or-one association (1008) indicates that A is associated with zero or one Bs; a "Has a" relationship (1010) indicates that A comprises a B and a C (e.g., where A is a form, and B and C are subforms); and an "Is a" relationship (1012) indicates that B is an A and C is an A (potentially used with forms and subforms).

Object Model diagrams, which comprise part of an object's documentation, are suitably used by the developer to determine the design elements contained in a given object. That is, all objects and applications are preferably created within a particular template of required fields, required objects, and required roles. Moreover, objects are created in accordance with a preferred element order (e.g., subform order).Each of these architectural aspects will now be described in turn.

On a large scale, all applications preferably conform to a structure which will best exploit an object-oriented methodology when used in the context of the various components of the present system. In a preferred embodiment, for example, all applications conform to the system-level object model set forth in **Figure 11**. Specifically, an application (or database) 1102 "has a" Configuration object 1104, All View object 1106, and Standard View Template 1108. Application 1102 also has one or more form objects 1110, wherein form object 1110 is a Response Form 1130, and has: zero or one Return objects 1120; zero or one Edit History objects 1122; zero or one Standard Action Buttons objects 1124; a Document ACL object 1126; and a Standard Subforms object 1128.

Configuration object 1104 preferably provides the standard configuration and setup mechanism for applications. It's main purpose is to allow developers to easily expose configuration controls to users, and to do so in a standard, modular way. Configuration object 1104 preferably comprises objects for storing configuration

settings related to the application as well as choices shown to users in keyword fields. Configuration object 1104 also preferably comprises a standard database configuration subform containing common settings, views for accessing configuration documents, and support for roles which control author access to documents.

All view object 1106 preferably provides a standard view for examining all documents in a database. This can be used by a user or developer for debugging purposes, or can be used in conjunction with script or HTML commands (e.g., LotusScript or Domino URLs) for programming purposes.

Standard View Template 1108 preferably comprises forms and action buttons designed to give a standard look and feel for all the views and search results pages in an application. This is particularly desirable in cases where the subject application is accessible to web clients.

Return object 1120 is preferably used to provide a return field to objects which call the form. The return field -- which may concatenate information received from multiple objects in a form -- may comprise information related to HTML code, an URL, a link back to the current document, or even background color of the returned page.

Edit History object 1122 is preferably used to retain a chronological list of editors for the particular document. In a preferred embodiment, Edit History object 1122 stores the edit history in a single field rather than a plurality of fields; this scheme eases the addition of new entries, provides flexibility with respect to the edit history data structure, and helps control disk space use.

Standard Action Buttons object 1124 preferably comprises a subform containing a number of common action buttons, for example, "Save", "Close", "Edit", "Delete", "Previous/Next", and E-mail feedback. The use of such a subform helps provide consistency across documents and across applications.

Document ACL object 1126 provides standardized control over Reader and Author fields needed in an object-oriented application. This object preferably accepts input from other objects and grants author access to all documents in accordance with specified access conditions (e.g., through the use of a standard role. Among other things, Document ACL object 1126 simplifies the task of troubleshooting with respect to access control problems.

Standard Subforms object 1128 preferably comprises a set of standard subforms, for example, a standard header subform (including a unique document ID, an author code, modification dates, and the like) and a standard footer subform.

In addition to required objects, development standards might include one or more required fields. Such fields might be used, for example, to communicate information to the user. In an illustrated embodiment, two fields are required for all applications: txt_DocSummary and txt_ResponseLine. The txt_DocSummary field suitably contains a one line summary of the document contents. Agents and views can use this field to describe the document to the user. The txt_ResponseLine field is suitably used on all response documents. It should contain a one line summary of the content of the response document. This field is used by views in response only columns.

Architectural standards might also specify the order of elements that appear in an object, particularly when a form comprises one or more subforms. For example, with reference to the various elements described above in connection with **Figure 11**, the following order of elements is suitable:

1. Standard Document Header subform
2. Standard Document Action Buttons
3. <Form-Specific elements>
4. Horizontal Divider element
5. Document ACL object
6. Edit History object
7. Standard Document Footer
8. Return object

In addition to required objects and required fields, a set of required roles may be specified. For example, in an illustrated embodiment, roles designated as "EditAll", "DBConfig", and "Keyword Config"are required. The "EditAll" role is suitably used in conjunction with Document ACL object 1126 to grant editor and reader rights to all documents in the database. The "DbConfig" role is suitably used to grant database-level rights to the application, and is advantageously used by any object that needs to restrict access to users who are owners or configurers of the

application. The "Keyword Config" role is suitably used to grant keyword setup rights to the application. The latter two roles are preferably assigned to users at the discretion of the developer/administrator/owner of the application.

Although the invention has been described herein in conjunction with the appended drawings, those skilled in the art will appreciate that the scope of the invention is not so limited. Modifications in the selection, design, and arrangement of the various components and steps discussed herein may be made without departing from the scope of the claims.

CLAIMS

1. A method for using a programmable digital computer to create object-oriented applications within a non-object-oriented software environment implemented on said programmable digital computer, wherein said non-object-oriented software environment is configured to allow creation of an application comprising one or more design elements, said method comprising the steps of:

creating, in accordance with a predetermined set of design standards, an object within said non-object-oriented software environment, wherein:

said object comprises at least one of said design elements;

said object is characterized by inbound public interfaces, outbound interfaces, and dependencies implemented using said design elements;

transferring said object to said application;

determining, for said object, a level of compliance to said predetermined set of design standards.

2. The method of claim 1, wherein said predetermined set of design standards comprises naming standards and architectural standards.

3. The method of claim 2, wherein said object has an object name associated therewith, and said naming standards specify that said name of said object comprises an element prefix, an object ID, and an element name.

4. The method of claim 3, wherein said naming standards specify that said name of said object further comprises an instance number.

5. The method of claim 2, wherein said architectual standards comprise a graphical representation of said design elements configured in accordance with an object modeling technique.

6. The method of claim 2, wherein said architectual standards specify an order in which said design elements should appear in said object.

7. The method of claim 2, wherein said architectual standards specify a set of required design elements for said object.

8. A system for developing an object-oriented application within a non-object-oriented software environment provided on a digital computer, said system comprising:

at least one object repository for storing objects;

an application builder configured to allow transfer of said objects from said object repository to said object-oriented application;

an application analyzer for analyzing said object-oriented application and determining a level of compliance to a predetermined set of design standards.

9. The system of claim 8, further comprising at least one carrier database, wherein said application builder is further configured to allow transfer of said objects from said carrier database to said repository.

10. The system of claim 9, further comprising a code fragment library.

11. The system of claim 8, wherein said predetermined set of design standards comprises naming standards and architectual standards associated with said objects.

12. The system of claim 11, wherein each of said objects have an object name associated therewith, and said naming standards specify that said object name comprises an element prefix, an object ID, and an element name.

13. The system of claim 12, wherein said naming standards specify that said object name further comprises an instance number.

14. The system of claim 11, wherein said architectual standards comprise a graphical representation of said objects configured in accordance with an object modeling technique.

15. The system of claim 11, wherein said architectual standards specify an order in which said design elements should appear in said object.

16. The method of claim 11, wherein said architectural standards specify a set of required design elements for said objects.

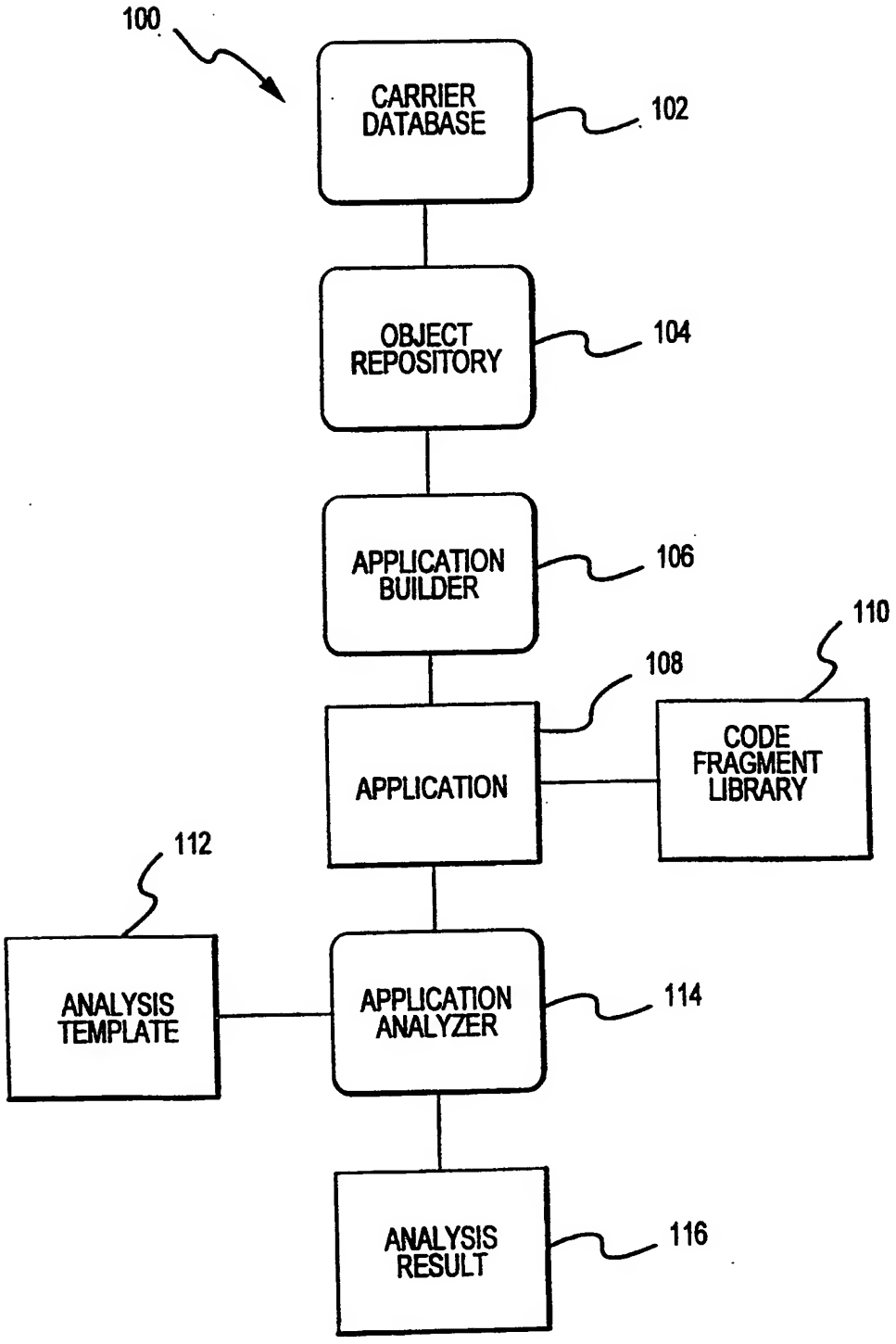

100
CARRIER DATABASE
102
OBJECT REPOSITORY
104
APPLICATION BUILDER
106
110
108
APPLICATION
CODE FRAGMENT LIBRARY
112
ANALYSIS TEMPLATE
APPLICATION ANALYZER
114
ANALYSIS RESULT
116

FIG.1

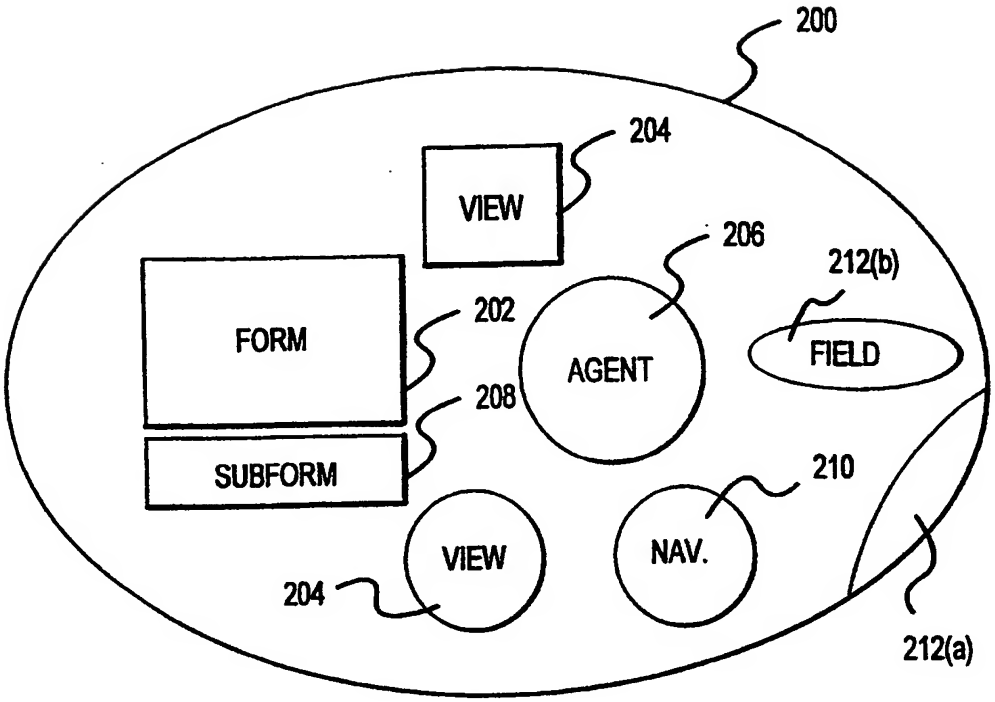

200
204
VIEW
206
212(b)
FORM
202
AGENT
FIELD
208
SUBFORM
210
VIEW
NAV.
204
212(a)

FIG.2

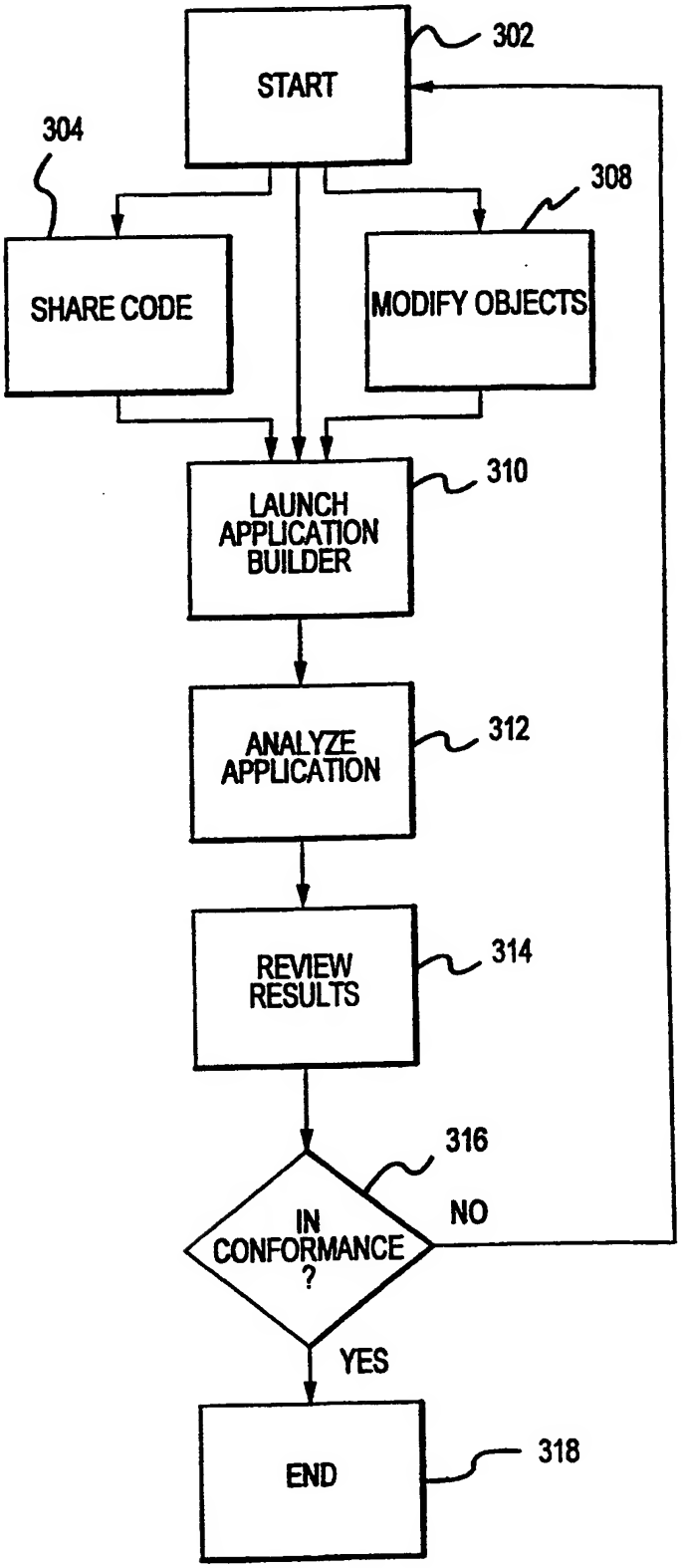

START
302
304
SHARE CODE
308
MODIFY OBJECTS
LAUNCH APPLICATION BUILDER
310
ANALYZE APPLICATION
312
REVIEW RESULTS
314
316
IN CONFORMANCE ?
NO
YES
END
318

FIG.3

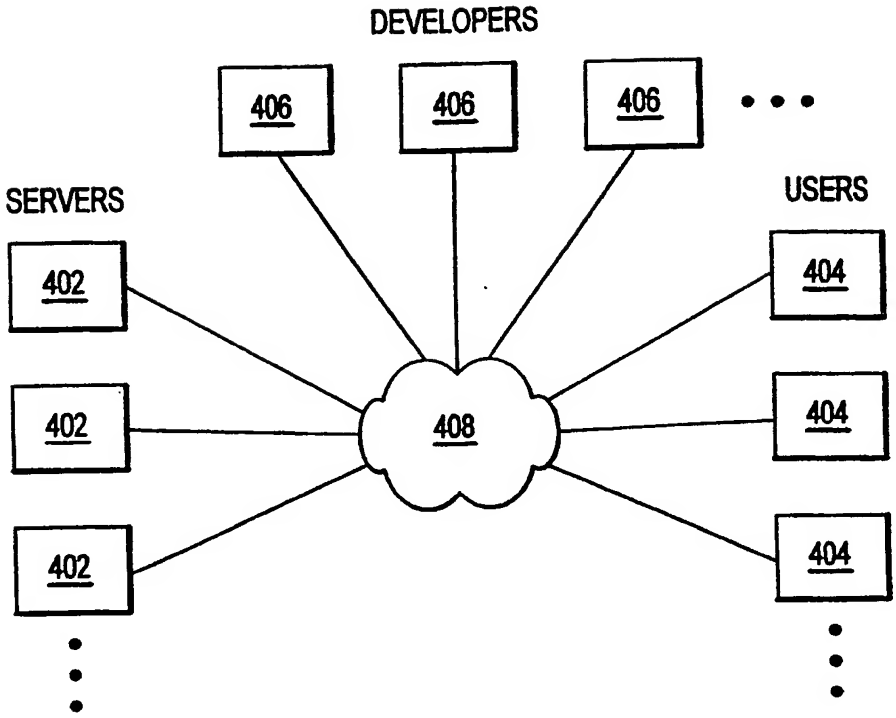

DEVELOPERS
406
406
406
SERVERS
USERS
402
402
402
404
404
404
408

FIG.4

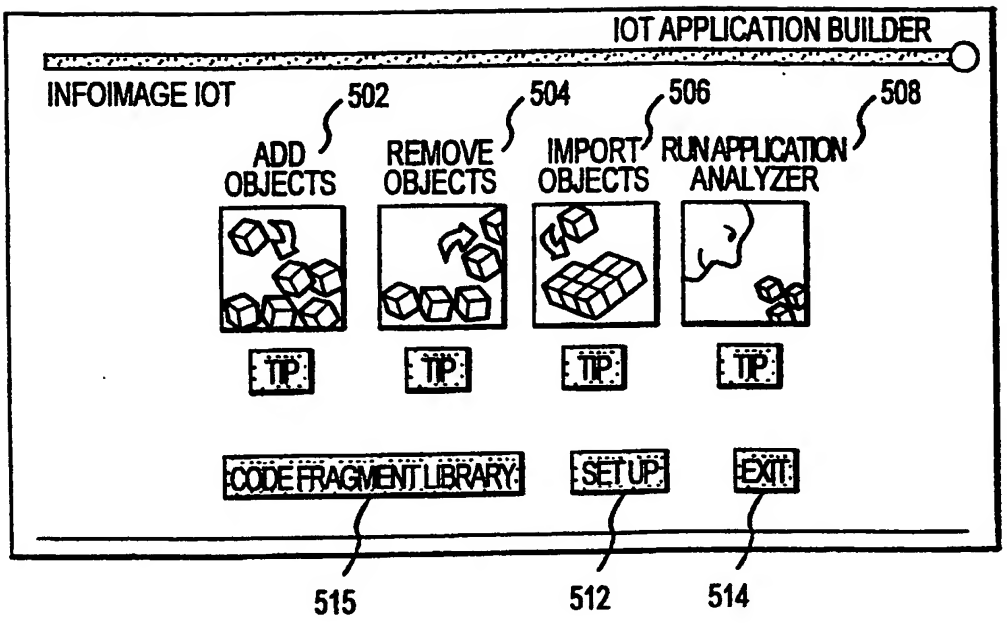

IOT APPLICATION BUILDER
INFOIMAGE IOT
502
504
506
508
ADD OBJECTS
REMOVE OBJECTS
IMPORT OBJECTS
RUN APPLICATION ANALYZER
TIP
TIP
TIP
TIP
CODE FRAGMENT LIBRARY
SET UP
EXIT
515
512
514

FIG.5

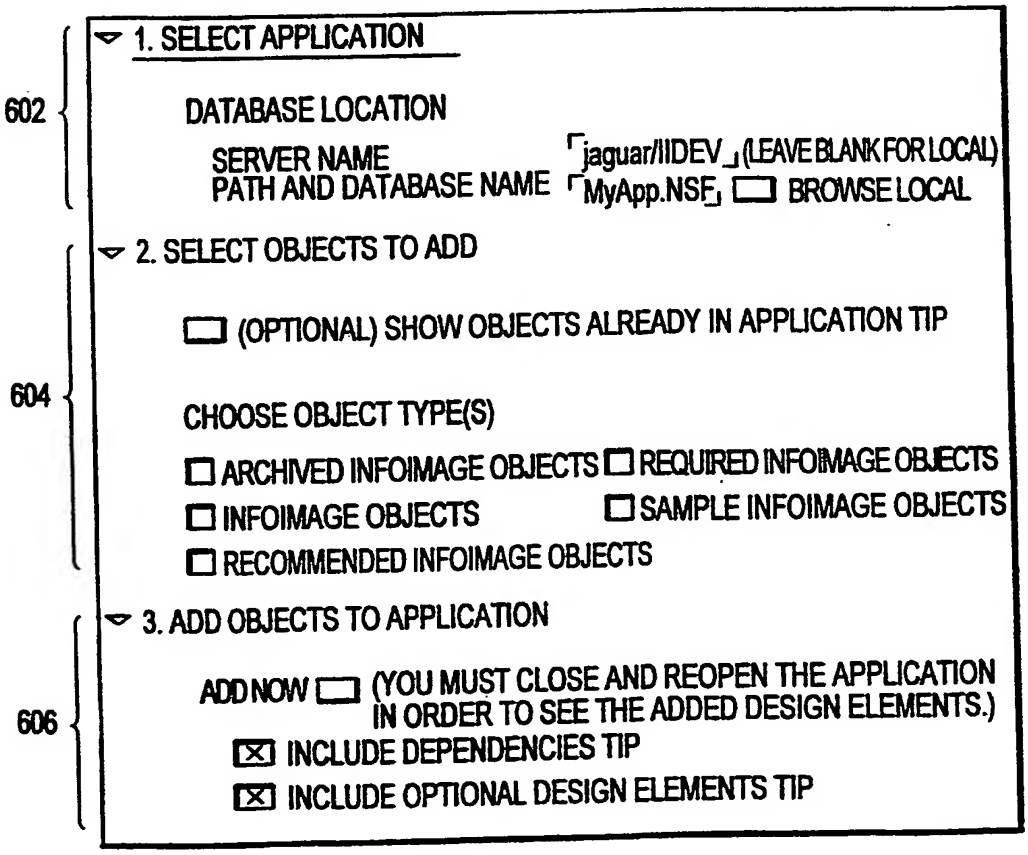

602
1. SELECT APPLICATION
DATABASE LOCATION
SERVER NAME
jaguar/IIDEV (LEAVE BLANK FOR LOCAL)
PATH AND DATABASE NAME
MyApp.NSF
BROWSE LOCAL
604
2. SELECT OBJECTS TO ADD
(OPTIONAL) SHOW OBJECTS ALREADY IN APPLICATION TIP
CHOOSE OBJECT TYPE(S)
ARCHIVED INFOIMAGE OBJECTS
REQUIRED INFOIMAGE OBJECTS
INFOIMAGE OBJECTS
SAMPLE INFOIMAGE OBJECTS
RECOMMENDED INFOIMAGE OBJECTS
606
3. ADD OBJECTS TO APPLICATION
ADD NOW
(YOU MUST CLOSE AND REOPEN THE APPLICATION IN ORDER TO SEE THE ADDED DESIGN ELEMENTS.)
INCLUDE DEPENDENCIES TIP
INCLUDE OPTIONAL DESIGN ELEMENTS TIP

FIG.6

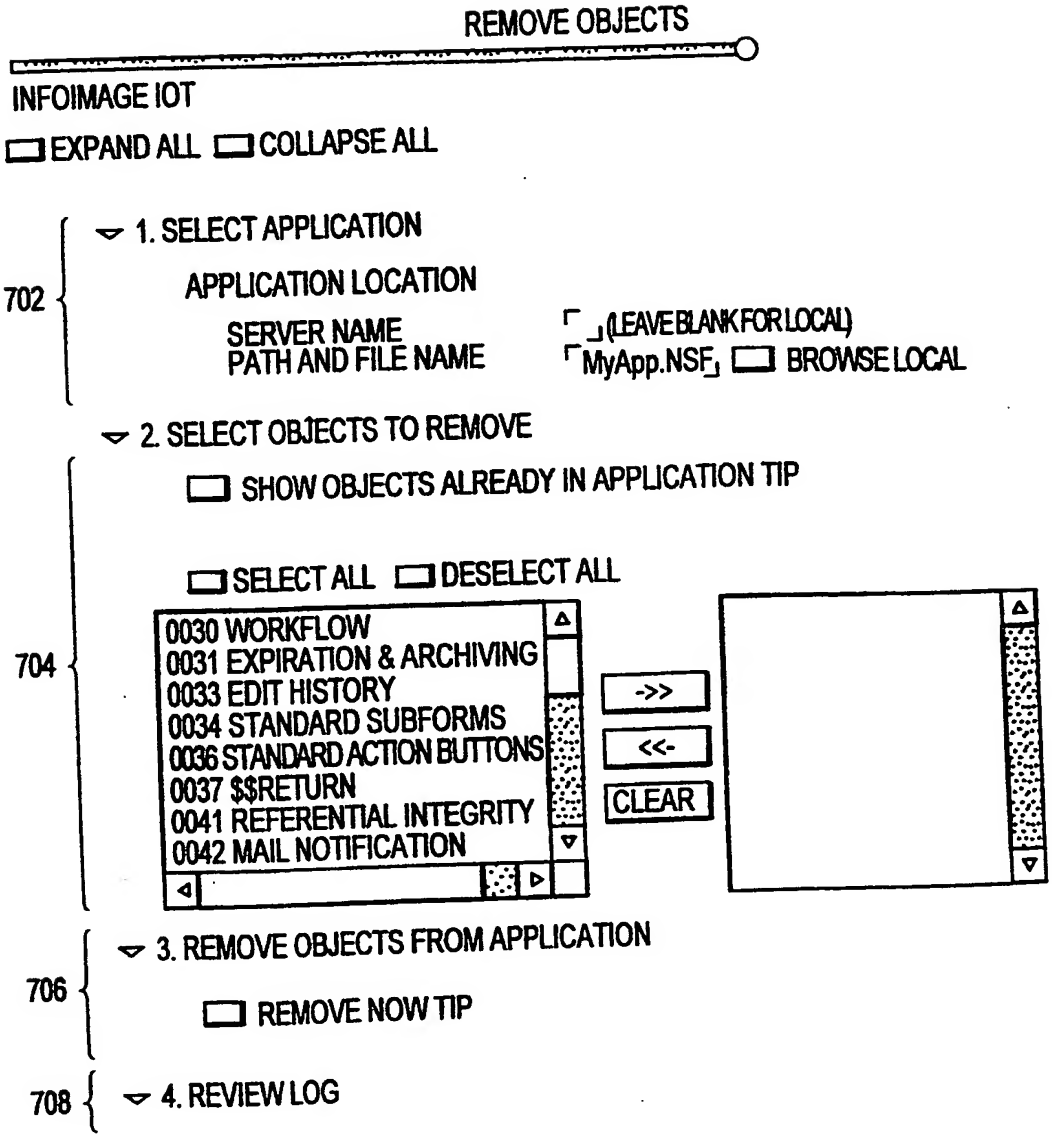

REMOVE OBJECTS
INFOIMAGE IOT
EXPAND ALL
COLLAPSE ALL
1. SELECT APPLICATION
APPLICATION LOCATION
SERVER NAME
(LEAVE BLANK FOR LOCAL)
PATH AND FILE NAME
MyApp.NSF
BROWSE LOCAL
2. SELECT OBJECTS TO REMOVE
SHOW OBJECTS ALREADY IN APPLICATION TIP
SELECT ALL
DESELECT ALL
0030 WORKFLOW
0031 EXPIRATION & ARCHIVING
0033 EDIT HISTORY
0034 STANDARD SUBFORMS
0036 STANDARD ACTION BUTTONS
0037 $$RETURN
0041 REFERENTIAL INTEGRITY
0042 MAIL NOTIFICATION
->>
<<-
CLEAR
3. REMOVE OBJECTS FROM APPLICATION
REMOVE NOW TIP
4. REVIEW LOG
702
704
706
708

FIG.7

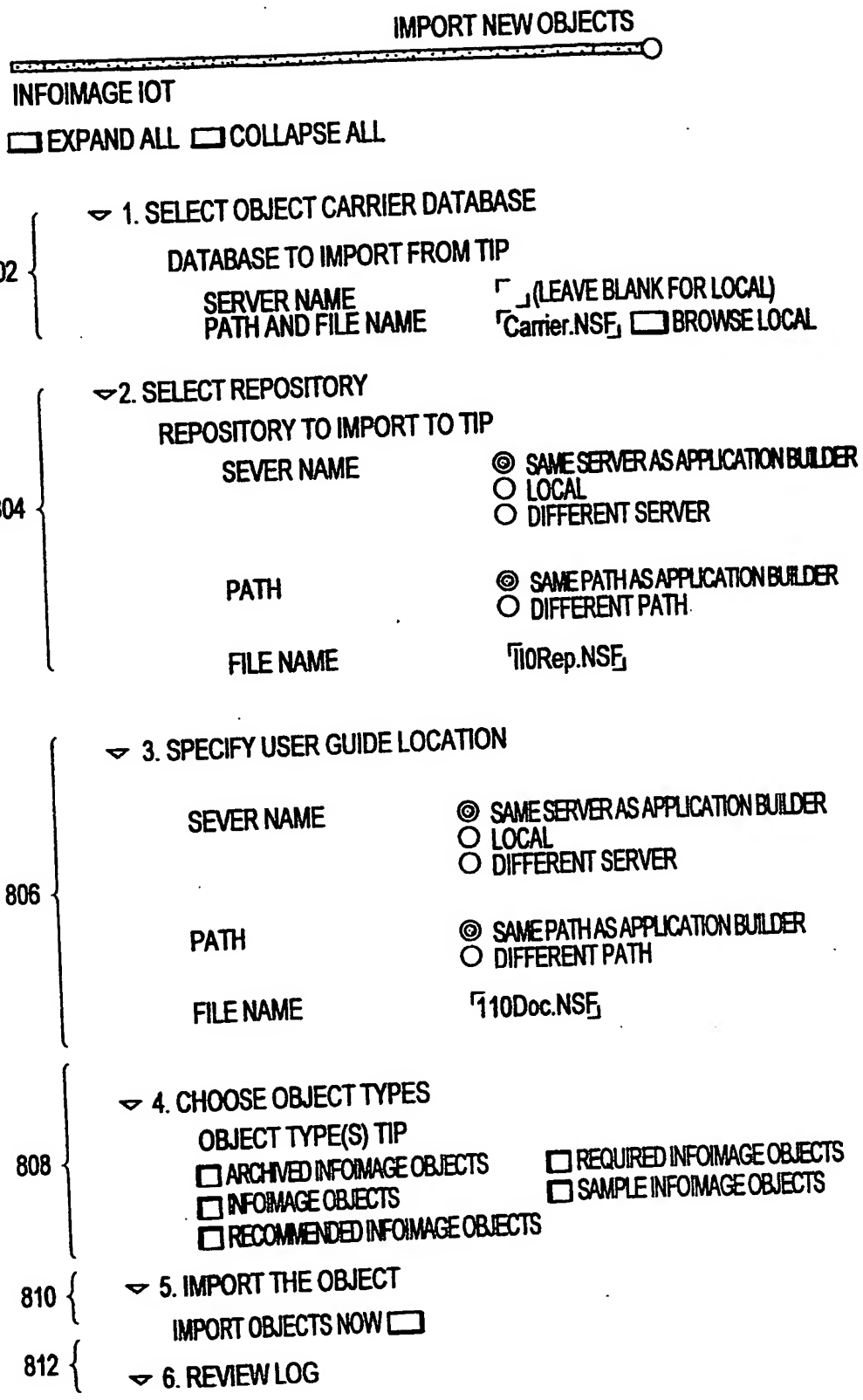

IMPORT NEW OBJECTS
INFOIMAGE IOT
EXPAND ALL COLLAPSE ALL
1. SELECT OBJECT CARRIER DATABASE
802
DATABASE TO IMPORT FROM TIP
SERVER NAME
(LEAVE BLANK FOR LOCAL)
PATH AND FILE NAME
Carrier.NSF
BROWSE LOCAL
2. SELECT REPOSITORY
804
REPOSITORY TO IMPORT TO TIP
SEVER NAME
SAME SERVER AS APPLICATION BUILDER
LOCAL
DIFFERENT SERVER
PATH
SAME PATH AS APPLICATION BUILDER
DIFFERENT PATH
FILE NAME
I10Rep.NSF
3. SPECIFY USER GUIDE LOCATION
806
SEVER NAME
SAME SERVER AS APPLICATION BUILDER
LOCAL
DIFFERENT SERVER
PATH
SAME PATH AS APPLICATION BUILDER
DIFFERENT PATH
FILE NAME
I10Doc.NSF
4. CHOOSE OBJECT TYPES
808
OBJECT TYPE(S) TIP
ARCHIVED INFOIMAGE OBJECTS
INFOIMAGE OBJECTS
RECOMMENDED INFOIMAGE OBJECTS
REQUIRED INFOIMAGE OBJECTS
SAMPLE INFOIMAGE OBJECTS
810
5. IMPORT THE OBJECT
IMPORT OBJECTS NOW
812
6. REVIEW LOG

FIG.8

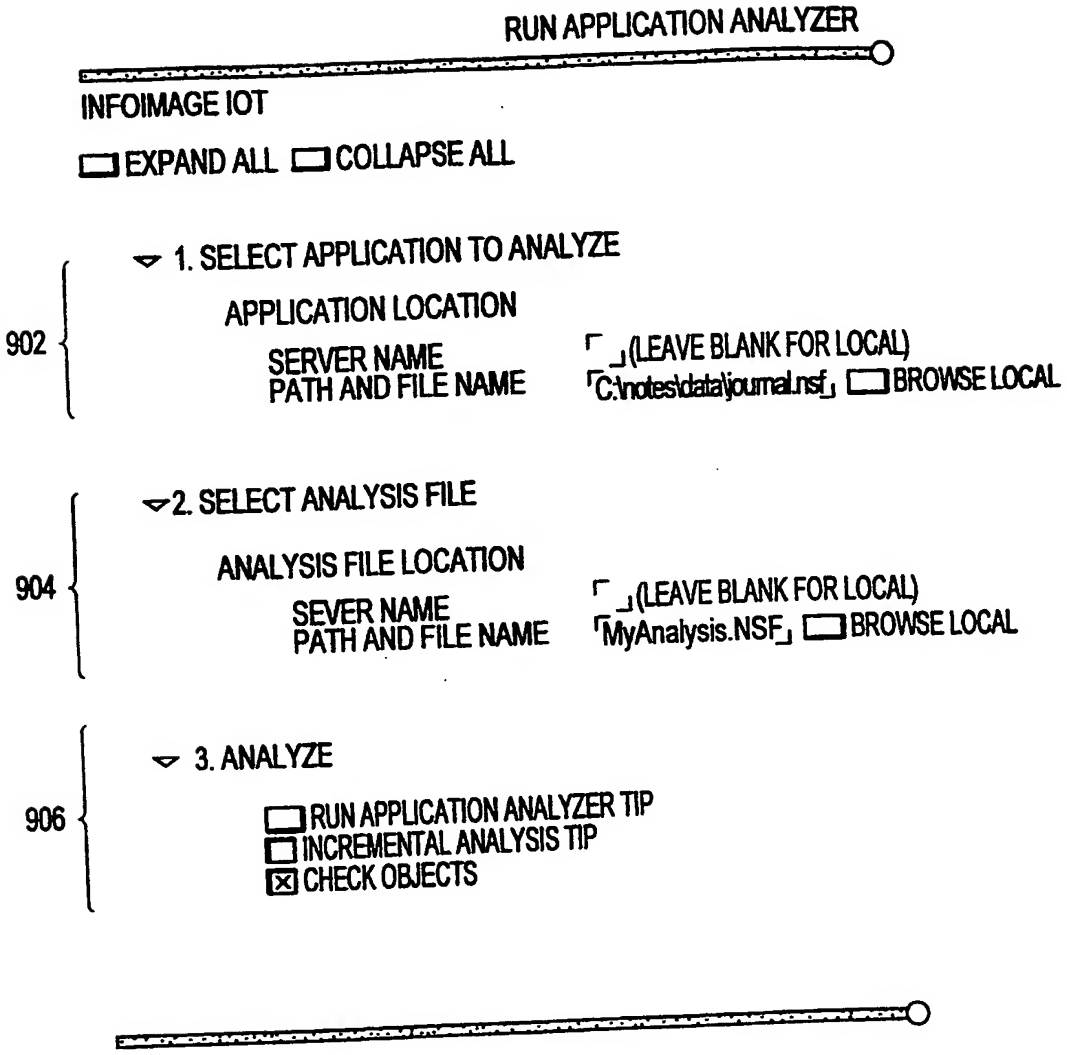

RUN APPLICATION ANALYZER
INFOIMAGE IOT
EXPAND ALL
COLLAPSE ALL
1. SELECT APPLICATION TO ANALYZE
APPLICATION LOCATION
SERVER NAME
PATH AND FILE NAME
(LEAVE BLANK FOR LOCAL)
C:\notes\data\journal.nsf
BROWSE LOCAL
902
2. SELECT ANALYSIS FILE
ANALYSIS FILE LOCATION
SEVER NAME
PATH AND FILE NAME
(LEAVE BLANK FOR LOCAL)
MyAnalysis.NSF
BROWSE LOCAL
904
3. ANALYZE
RUN APPLICATION ANALYZER TIP
INCREMENTAL ANALYSIS TIP
CHECK OBJECTS
906

FIG.9

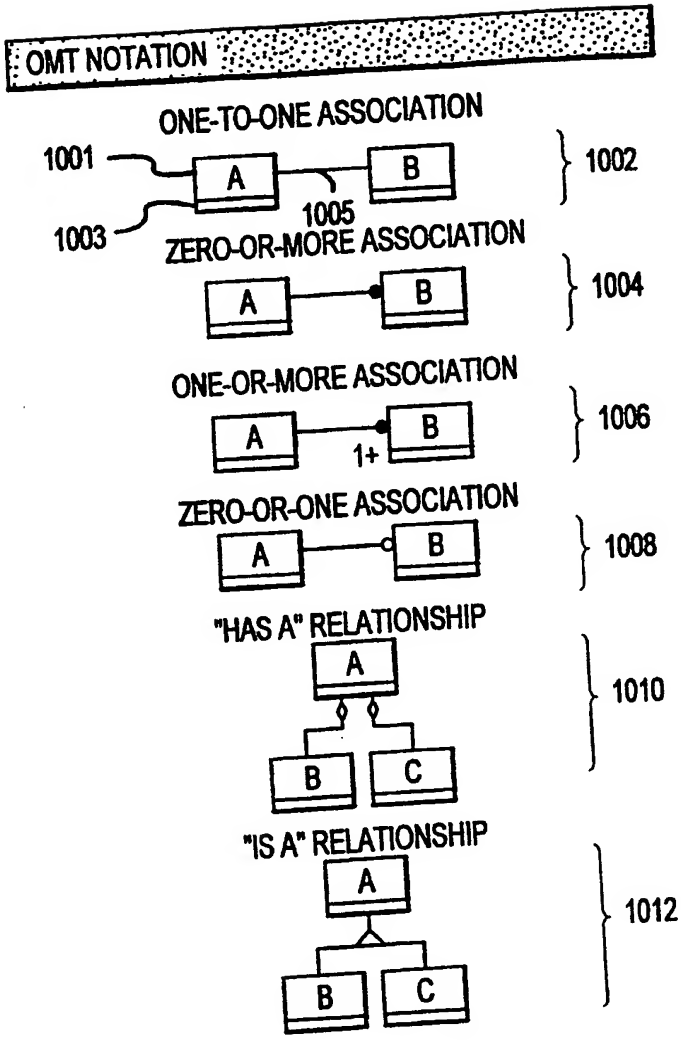

OMT NOTATION
ONE-TO-ONE ASSOCIATION
1001
A
B
1002
1005
1003
ZERO-OR-MORE ASSOCIATION
A
B
1004
ONE-OR-MORE ASSOCIATION
A
1+
B
1006
ZERO-OR-ONE ASSOCIATION
A
B
1008
"HAS A" RELATIONSHIP
A
B
C
1010
"IS A" RELATIONSHIP
A
B
C
1012

FIG.10

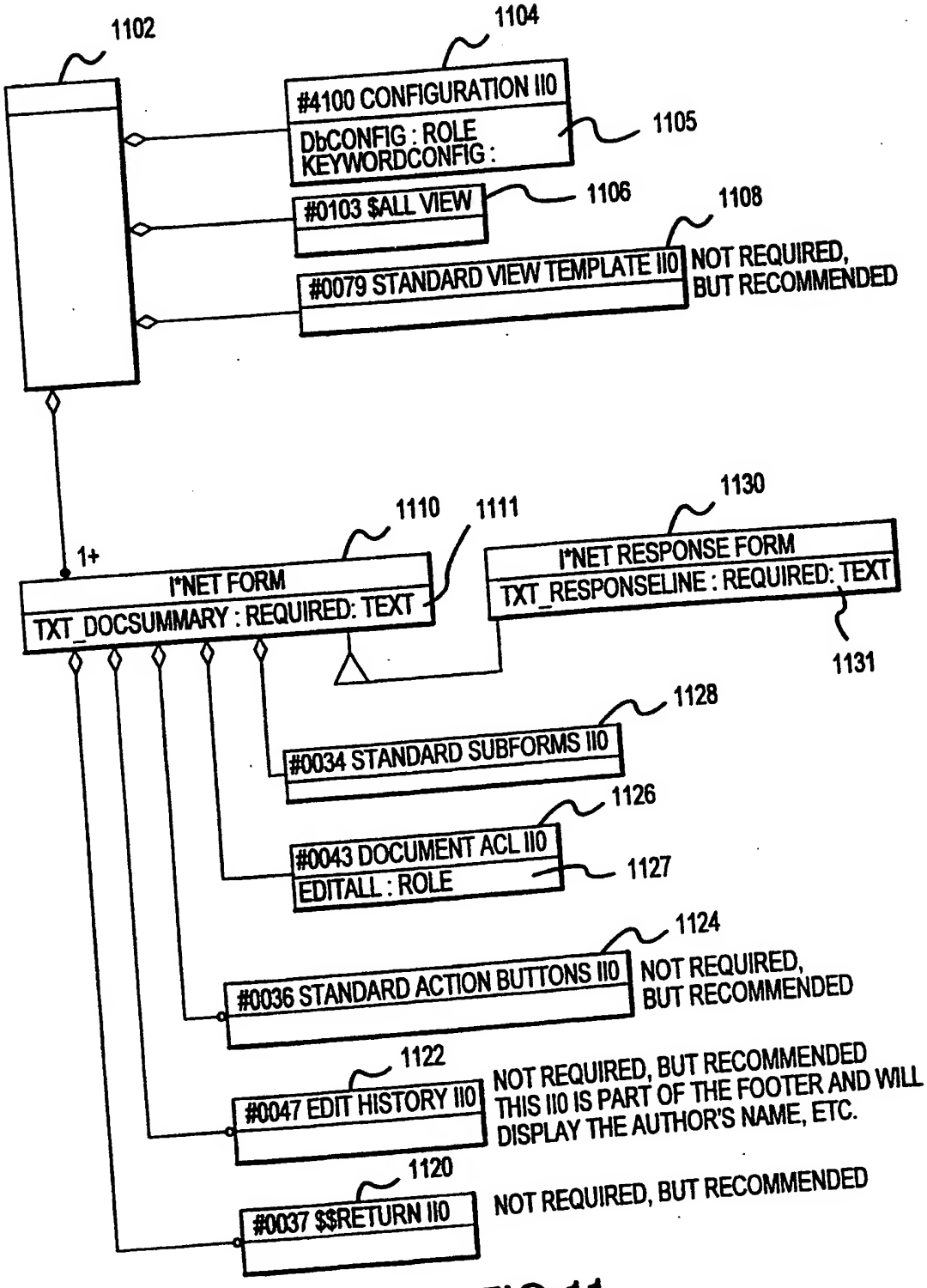

1102
1104
#4100 CONFIGURATION II0
DbCONFIG : ROLE
KEYWORDCONFIG :
1105
#0103 $ALL VIEW
1106
1108
#0079 STANDARD VIEW TEMPLATE II0
NOT REQUIRED, BUT RECOMMENDED
1+
1110
1111
1130
I*NET FORM
TXT_DOCSUMMARY : REQUIRED: TEXT
I*NET RESPONSE FORM
TXT_RESPONSELINE : REQUIRED: TEXT
1131
1128
#0034 STANDARD SUBFORMS II0
1126
#0043 DOCUMENT ACL II0
EDITALL : ROLE
1127
1124
#0036 STANDARD ACTION BUTTONS II0
NOT REQUIRED, BUT RECOMMENDED
1122
#0047 EDIT HISTORY II0
NOT REQUIRED, BUT RECOMMENDED
THIS II0 IS PART OF THE FOOTER AND WILL DISPLAY THE AUTHOR'S NAME, ETC.
1120
#0037 $$RETURN II0
NOT REQUIRED, BUT RECOMMENDED

FIG.11

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US99/06126

A. CLASSIFICATION OF SUBJECT MATTER

IPC(6) :G06F 15/20
US CL : 395/705, 703, 702

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)

U.S. : 395/705, 703, 702; 709/303; 707/103

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)

APS, OOPSLA, IEEE

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
Y	US 5,325,533 A (MCINERNEY et al) 28 June 1994, col.1, lines 6-23; col. 2, line 66 - col. 5, line 49; col. 9, lines 25-40; col. 12, line 44 - col. 13, line 51; col. 16, lines 24-66; col. 22, lines 19-41; col. 25, lines 26-40.	1-2, 5-7, 8-11, 14-16
Y	US 5,542,078 A (MARTEL et al) 30 July 1996, col. 3, line 35 - col. 4, line 10; col. 58, lines 32-35; figures 2, 6.	1-2, 5-7, 8-11, 14-16
Y	US 5,509,116 A (HIRAGA et al) 16 April 1996, figures 14, 15.	3-4, 12-13
Y	US 5,497,491 A (MITCHELL et al) 05 March 1996, figures 5, 8B, 9; col. 2, lines 26-48.	3-4, 12-13

[X] Further documents are listed in the continuation of Box C. [ ] See patent family annex.

* Special categories of cited documents.

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier document published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art

"&" document member of the same patent family

Date of the actual completion of the international search
09 JUNE 1999

Date of mailing of the international search report
20 AUG 1999

Name and mailing address of the ISA/US
Commissioner of Patents and Trademarks
Box PCT
Washington, D.C. 20231
Facsimile No. (703) 305-3230

Authorized officer
SUE LAO
Joni Hill
Telephone No. (703) 305-9657

Form PCT/ISA/210 (second sheet)(July 1992)*